

Package ‘jekylldown’

October 14, 2026

Title Create, Build and Serve 'Jekyll' Websites from 'R Markdown' and 'Quarto'

Version 0.3.4

Description Do for 'Jekyll' what 'blogdown' does for 'Hugo': create sites from the most popular themes ('al-folio', 'Chirpy', 'Minimal Mistakes', 'minima'), write posts in R Markdown or 'Quarto' that are knitted to Markdown consumed natively by 'Jekyll', customize theme colors, fonts and elements declaratively from R, migrate 'blogdown'/'Hugo' sites, build and serve the site locally with live reload, and diagnose the Ruby/'Jekyll' toolchain. 'Jekyll' is invoked strictly via its command line interface; no R-Ruby language bridge is used.

License MIT + file LICENSE

Encoding UTF-8

Language en-US

Imports cli, fs, knitr, processx, servr, xfun, yaml

Suggests archive, quarto, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

SystemRequirements Ruby (>= 3.0), jekyll, bundler

URL <https://github.com/allanvc/jekylldown>

BugReports <https://github.com/allanvc/jekylldown/issues>

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Allan Quadros [aut, cre]

Maintainer Allan Quadros <allanvcq@gmail.com>

Repository CRAN

Date/Publication 2026-10-14 11:20:08 UTC

Contents

add_css	2
add_feed	3
add_footer_credit	4
add_mathjax	5
build_site	6
bundle_install	7
check	8
install_git	8
install_quarto	9
install_ruby	10
knit_post	11
migrate_hugo	12
new_post	14
new_site	15
serve_site	17
set_element_style	18
set_theme_color	19
set_theme_font	20
set_theme_skin	22
set_theme_style	22
stop_server	23
use_pages_workflow	24
use_r_bloggers	25
Index	26

add_css	<i>Add (or remove) a managed block of custom CSS</i>
---------	--

Description

The escape hatch under `set_theme_style()`, `set_theme_font()` and `set_element_style()`: free-form CSS, but written into a marker-delimited block in the site-local stylesheet, so it is idempotent per id, easy to find, and removable – instead of hand edits scattered through theme files. Works with any theme whose site has (or can get) a local `main.scss`.

Usage

```
add_css(css, id = "custom", dir)
```

Arguments

css	Character vector of CSS lines. <code>character(0)</code> (or <code>""</code>) removes the block with this id.
id	Identifier of the block (lowercase letters, digits, <code>-</code> , <code>_</code>), so independent customizations do not overwrite each other.

`dir` Site root, or any directory inside it. Like `build_site()`, the function climbs to the enclosing site.

Value

Invisibly, the path of the stylesheet written.

Examples

```
site <- new_site(tempfile("my-blog"))
add_css(".profile img { border-radius: 50%; }", id = "round-avatar",
        dir = site)
# remove it later
add_css(character(0), id = "round-avatar", dir = site)
unlink(site, recursive = TRUE)
```

add_feed	<i>Atom feeds with a correct title, optionally per category</i>
----------	---

Description

Writes a site-level copy of jekyll-feed's template to `_includes/atom-feed.xml` and the feed pages that render it: `feed.xml` with every post, plus `feed/<category>.xml` for each category. A category feed carries the full text of the posts whose categories front-matter entry contains that value. The match is verbatim, so "R" and "r" are different categories. Aggregators such as R-Bloggers require exactly that kind of feed; see [use_r_bloggers\(\)](#) for the one-call setup.

Usage

```
add_feed(category = NULL, dir, force = FALSE)
```

Arguments

<code>category</code>	Optional character vector of post categories, one feed each at <code>feed/<category>.xml</code> . NULL (the default) writes the main feed only.
<code>dir</code>	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.
<code>force</code>	Regenerate <code>_includes/atom-feed.xml</code> even when it is current.

Details

The copy differs from the plugin's template in one place. al-folio sets `title: blank` in `_config.yml` to mean "build the site title from the author's name". The theme's layouts follow that convention, but jekyll-feed does not, and prints the literal word "blank" as the feed title. The copy handles that case and leaves the title of every other theme as it is.

The template is taken from the jekyll-feed gem installed for the site, at the version pinned in `Gemfile.lock`. That is the version the local build and the deploy workflow render with, so the

include follows the plugin instead of freezing a copy inside jekylldown. When the gem is not installed yet, the template of that version is fetched from GitHub. Offline, the copy shipped with the package is used. It is a complete template, and the feed pages need no plugin at all. The include records where it came from. Running the function again after a gem upgrade regenerates it from the new version, and `force = TRUE` regenerates it unconditionally. An `atom-feed.xml` you wrote yourself, without the jekylldown marker, is never touched.

Chirpy ships a main feed of its own at the same address, and Jekyll keeps the theme's version of `feed.xml` on that theme. The per-category feeds are jekylldown's on every theme.

In the feeds, root-relative image and link paths in the post body (`/assets/img/...`, which is what `knit_post()` writes) are made absolute from `url:` in `_config.yml`. Feed readers are supposed to resolve such paths against the entry's `xml:base`, but aggregators such as R-Bloggers ask for absolute URLs, so the feed does not depend on the reader.

`new_site()` and `migrate_hugo()` already call this on al-folio sites. Call it yourself on an existing site, or on any theme where you want per-category feeds.

Value

Invisibly, the paths of the feed pages written or confirmed (the main feed first), with the include's path as attribute "include".

See Also

`use_r_bloggers()`

Examples

```
site <- new_site(tempfile("my-blog"))
add_feed(category = "R", dir = site) # feed.xml plus feed/R.xml
readLines(file.path(site, "feed", "R.xml"))
unlink(site, recursive = TRUE)
```

add_footer_credit	<i>Add a jekylldown credit to the site footer</i>
-------------------	---

Description

Inserts "Built from R with jekylldown X.Y.Z." (with a link) into the footer line the theme already has – right after its "Powered by Jekyll ..." sentence on al-folio, and at the natural end of the footer on minima, Chirpy and Minimal Mistakes. `new_site()` adds the credit to fresh sites; call this yourself on migrated or existing ones. `build_site()` refreshes the version number automatically, so the footer follows package upgrades with no manual edits.

Usage

```
add_footer_credit(dir)

remove_footer_credit(dir)
```

Arguments

dir	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.
-----	--

Details

Per theme: on al-folio the sentence goes into the `footer_text:` entry of `_config.yml` (plain text you can reorder or edit). On the other themes the theme's `_includes/footer.html` is copied into the site (a local include shadows the gem's; for Chirpy and Minimal Mistakes the gem must be installed – run `bundle_install()` first) and the credit is added as a marker-delimited block.

Undo with `remove_footer_credit()`.

Value

Invisibly, the file the credit was written to.

Examples

```
## Not run:
add_footer_credit("my-site")
remove_footer_credit("my-site")

## End(Not run)
```

add_mathjax

Enable LaTeX math rendering (MathJax)

Description

R Markdown writers use LaTeX math, but not every Jekyll theme renders it out of the box (on minima, `$$...$$` shows up as literal text). This turns math on the way each theme expects:

Usage

```
add_mathjax(dir)
```

Arguments

dir	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.
-----	--

Details

- **al-folio**: sets its native `enable_math`: `true`;
- **Chirpy**: enables its per-page math flag site-wide through a front matter default;
- **Minimal Mistakes**: loads MathJax through the theme's `head_scripts` hook;
- **minima** (and unknown themes): copies the theme's `_includes/head.html` into the site and inserts the MathJax `<script>` before `</head>`, in a managed, idempotent block.

kramdown (Jekyll's Markdown engine) emits MathJax-v2-flavored output for math, so the v2 bundle is loaded where jekylldown injects the script itself; al-folio and Chirpy ship their own MathJax setups.

Value

Invisibly, a short description of what was changed.

Examples

```
## Not run:
add_mathjax("my-site")

## End(Not run)
```

build_site	<i>Knit sources and build the site</i>
------------	--

Description

Knits every outdated `_source/*.Rmd` and renders every outdated `_source/*.qmd` (Quarto) to `_posts/*.md` (see [knit_post\(\)](#)), then runs `jekyll build` if a usable Jekyll is available.

Usage

```
build_site(dir, local_jekyll = NULL, force = FALSE)
```

Arguments

<code>dir</code>	Directory in (or under) the site.
<code>local_jekyll</code>	Run <code>jekyll build</code> after knitting? <code>NULL</code> (default) auto-detects: builds when a <code>jekyll</code> executable is found, otherwise skips with a note (knitting alone is enough when GitHub Pages/Actions does the Jekyll build remotely).
<code>force</code>	Re-knit sources even when the output is up to date?

Value

Invisibly, the paths of the posts (re)knitted.

Examples

```
# knit only, no Jekyll needed: what a GitHub Actions setup uses
site <- new_site(tempfile("my-blog"))
build_site(site, local_jekyll = FALSE)
list.files(file.path(site, "_posts"))
unlink(site, recursive = TRUE)

## Not run:
# knit/render outdated posts, then build with the local Jekyll
build_site("my-site")

# re-knit everything from scratch
build_site("my-site", force = TRUE)

## End(Not run)
```

bundle_install*Install a site's gem dependencies with Bundler*

Description

Runs `bundle install` in the site root with the package's isolated gem environment injected, so a theme's Gemfile (al-folio has one) resolves without any manual `GEM_HOME/PATH` setup. Needed once per site (and after editing the Gemfile); afterwards `build_site()` and `serve_site()` automatically switch to `bundle exec jekyll`.

Usage

```
bundle_install(dir)
```

Arguments

`dir` Directory in (or under) the site.

Details

One external requirement: a Gemfile that installs gems straight from git repositories (al-folio's does) needs `git`, because Bundler shells out to it – the only step in the jekyll-down workflow that requires `git`. On Windows, `install_git()` puts a portable copy in the isolated toolchain; elsewhere install it with your package manager.

Value

Invisibly, the `processx::run()` result.

 check

Diagnose the Jekyll toolchain

Description

Prints a situation report on everything jekylldown needs: Ruby, gem, bundler and jekyll executables (looking first in the package's isolated GEM_HOME, then on the PATH), plus the current site, if any.

Usage

```
check(dir = ".")
```

Arguments

dir Directory in (or under) a Jekyll site. Defaults to the working directory; not finding a site is reported, not an error.

Value

Invisibly, a named list with the collected information.

Examples

```
jekylldown::check()
```

 install_git

Install a portable git into jekylldown's isolated toolchain (Windows)

Description

The one jekylldown step that needs git is `bundle_install()` on a Gemfile that pulls gems straight from git repositories (al-folio's does) – Bundler shells out to git for those. This function covers machines without git: it downloads MinGit, the portable build that Git for Windows publishes exactly for embedding (a plain .zip, no installer, no registry entries, no admin rights), and unpacks it under `tools::R_user_dir("jekylldown", "data")/git`, where every jekylldown-run Ruby command finds it automatically. Nothing is installed system-wide; delete the data directory to uninstall.

Usage

```
install_git(file = NULL, version = NULL, force = FALSE)
```

Arguments

file	Optional path to an already-downloaded MinGit-<version>-64-bit.zip, for offline installs. Default NULL downloads the latest release from GitHub.
version	Optional Git for Windows release to install instead of the latest, in asset notation (e.g. "2.55.0.3" for the tag v2.55.0.windows.3, "2.47.1" for v2.47.1.windows.1).
force	Re-download even when the toolchain directory already holds a working git? Default FALSE.

Details

Like `install_ruby()`, the release lookup uses the GitHub API and falls back to the releases/latest page on `github.com` (then to a pinned known-good release) on networks where `api.github.com` is unreachable. x86-64 only.

This is a private copy for the toolchain: it does not appear on your system PATH, so it does not replace a real git installation for everyday version-control work (for publishing a site with `git push`, install git from <https://git-scm.com/downloads/>).

Value

The path to the installed git executable, invisibly.

install_quarto	<i>Install the Quarto CLI into jekylldown's isolated toolchain</i>
----------------	--

Description

Downloads the latest Quarto release and unpacks it under `tools::R_user_dir("jekylldown", "data")/quarto`, where `build_site()`, `serve_site()` and `check()` find it automatically. Nothing is installed system-wide and no shell configuration is touched; uninstall by deleting that directory. Only needed to write posts in Quarto (`.qmd`) – see `new_post()`.

Usage

```
install_quarto(file = NULL)
```

Arguments

file	Optional path to an already-downloaded quarto-<version>-linux-amd64.tar.gz, for offline installs. Default NULL downloads the latest release from GitHub.
------	--

Details

Linux x86_64 only for now (the platform jekylldown's isolated toolchain targets); on other platforms install Quarto from <https://quarto.org/docs/get-started/> and it is picked up from the PATH.

Value

The path to the installed quar to executable, invisibly.

install_ruby	<i>Install Ruby and Jekyll into jekyll-down's isolated toolchain (Windows)</i>
--------------	--

Description

Windows has no system package manager to get Ruby from, so this function does the whole setup: it downloads the portable RubyInstaller archive (a .7z, no graphical installer, no registry entries), unpacks it under `tools\R_user_dir("jekyll-down", "data")/ruby`, and installs the **jekyll**, **bundler** and **minima** (the default theme) gems into it. Nothing is installed system-wide and no shell configuration is touched; uninstall by deleting the data directory. `build_site()`, `serve_site()` and `check()` find this toolchain automatically.

Usage

```
install_ruby(file = NULL, devkit = TRUE, version = NULL, force = FALSE)
```

Arguments

file	Optional path to an already-downloaded rubyinstaller-<version>-x64.7z, for offline installs. Default NULL downloads the latest release from GitHub. (The release lookup uses the GitHub API; on networks where <code>api.github.com</code> is unreachable, a known-good pinned release is downloaded directly from <code>github.com</code> instead.)
devkit	Set up the MSYS2 toolchain (via <code>ridk install</code>), which gems with C extensions – including Jekyll's own dependency 'eventmachine' – need at install time? Default TRUE.
version	Optional RubyInstaller release to install instead of the latest, as tagged upstream (e.g. "3.3.7-1").
force	Re-download Ruby even when the toolchain directory already holds a working one? Default FALSE: re-running the function reuses the provisioned Ruby and only completes what is missing (MSYS2 pieces, gems – already-installed gems are skipped).

Details

Unpacking the .7z archive needs the **archive** package (`install.packages("archive")`).

The MSYS2 build tools are set up by default (`devkit = TRUE`, via RubyInstaller's `ridk install`): Jekyll depends on the 'eventmachine' gem, which has no precompiled binary for current Rubies and must be compiled at install time. `devkit = FALSE` skips that large download for the rare setups that already have a working MSYS2 toolchain.

On Linux and macOS install Ruby with your package manager instead (see the README's prerequisites); jekyll-down picks it up from the PATH.

Value

The path to the installed `jeekyll` command, invisibly.

knit_post	<i>Knit one source post to Jekyll-ready Markdown</i>
-----------	--

Description

Knits a single `.Rmd` with the conventions inherited from Yihui Xie's `knitr-jeekyll`: front matter preserved, output fenced for `kramdown/rouge` via `knitr::render_jeekyll()`, figures written to `assets/img/posts/<post-name>/` and referenced through `{{ site.baseurl }}` so they resolve wherever the site is mounted.

Usage

```
knit_post(input, output, root = NULL, method = NULL)
```

Arguments

input	Path to the <code>.Rmd</code> (or <code>.qmd</code>) source.
output	Path of the <code>.md</code> to write.
root	Site root; found from input when <code>NULL</code> .
method	For <code>.Rmd</code> sources: <code>"knitr"</code> (default) knits with plain <code>knitr</code> – fast, no <code>pandoc</code> needed; <code>"pandoc"</code> renders through <code>rmarkdown/pandoc</code> , enabling citations (bibliography:), footnotes and cross-references. <code>NULL</code> (the default) takes the post's <code>knit_method</code> : front matter key if present, then <code>options(jekyll::knit_method =)</code> , then <code>"knitr"</code> . Ignored for <code>.qmd</code> (always the Quarto CLI).

Details

A `.qmd` input is rendered with the Quarto CLI (`quarto render --to gfm`) instead, and the result is adapted to the same conventions: the front matter of the `.qmd` is carried over (minus Quarto-only keys, with `layout: post` added), and figures are moved to `assets/img/posts/<post-name>/`. Quarto is looked up in `jekyll::down`'s isolated toolchain (see `install_quarto()`), then on the `PATH`; `options(jekyll::quarto = "/path/to/quarto")` overrides both.

In both cases Markdown pipe tables get a `{: .table}` `kramdown` IAL so the theme styles them (al-folio only styles Bootstrap's `.table` class; a bare `<table>` looks unstyled).

Mostly called for you by `build_site()` and `serve_site()` (the latter via the generated `build.R`), but exported so those entry points and users can share one implementation.

Value

output, invisibly.

Examples

```
# normally called for you by build_site()/serve_site()
site <- new_site(tempfile("my-blog"))
src <- list.files(file.path(site, "_source"), full.names = TRUE)[1]
out <- file.path(site, "_posts", xfun::with_ext(basename(src), "md"))
knit_post(src, out)
readLines(out, n = 8)
unlink(site, recursive = TRUE)

## Not run:
# force the pandoc pipeline for one post (citations, footnotes, ...)
knit_post("my-site/_source/2026-01-01-paper.Rmd",
          "my-site/_posts/2026-01-01-paper.md", method = "pandoc")

## End(Not run)
```

migrate_hugo

Migrate a Hugo/blogdown site to Jekyll (best effort)

Description

Creates a fresh Jekyll site with `new_site()` and converts the mechanical parts of a Hugo site into it. The Hugo config (config.toml/.yaml) is read first and drives the migration:

Usage

```
migrate_hugo(
  from,
  to,
  theme = c("minima", "al-folio", "chirpy", "minimal-mistakes"),
  drafts = TRUE,
  only_referenced = FALSE,
  publications = c("bib", "html"),
  theme_color = NULL,
  rerender = TRUE
)
```

Arguments

from	Root of the existing Hugo site (must contain content/).
to	Directory to create the Jekyll site in (must not exist or be empty).
theme	Passed to <code>new_site()</code> .
drafts	Migrate draft posts (as published: false)? Default TRUE.
only_referenced	Copy from static/ only the files referenced by the migrated content (posts, pages, config)? Default FALSE (copy everything, as Hugo serves it).

publications	How to migrate a publications page found in the menu (slug publications, papers or pubs). "bib" (default) extracts the DOIs from the page, fetches each reference as BibTeX from https://www.doi.org/ (Crossref/DataCite content negotiation; needs network), pairs preview images found next to each DOI, and writes <code>_bibliography/papers.bib</code> so al-folio's jekyll-scholar renders the publications page with the theme's badges and buttons. Entries without a DOI are reported for manual addition. On themes without jekyll-scholar (minima, Chirpy, Minimal Mistakes) the page is rebuilt instead as a clean Markdown list of formatted citations (APA, also from doi.org) – raw-HTML publication pages rarely survive kramdown. "html" keeps your page as-is; "bib" falls back to it automatically when the page has no DOI or there is no network.
theme_color	Accent color for al-folio, passed to <code>set_theme_color()</code> (a palette name like "red" or a #hex value). Applied right away when the theme gems are already installed; otherwise remembered in the report as a step to run after <code>bundle_install()</code> . Default NULL keeps the theme default.
rerender	Should <code>.Rmd/.Rmarkdown</code> sources be migrated to <code>_source/</code> for re-knitting (TRUE, default)? With FALSE, when a post's committed rendered output (<code>.markdown/.md</code>) sits next to its R source, the rendered file is migrated as a plain post instead – the right choice for archival migrations, where years of old posts should not have their code executed again.

Details

- posts under `content/post/`, `content/posts/` or `content/blog/` are renamed to Jekyll's YYYY-MM-DD-slug convention; `.Rmd/.Rmarkdown` sources go to `_source/` (to be knitted by jekylldown), plain `.md/.markdown` to `_posts/`;
- every page referenced by the Hugo menu (`[[menu.main]]`) is migrated to the theme's own convention, with permalink preserving the Hugo URL and the menu order kept: al-folio gets `_pages/<slug>.md` with `nav_order`, Chirpy gets `_tabs/<slug>.md` with `order`, Minimal Mistakes gets `_pages/<slug>.md` plus an entry in `_data/navigation.yml`, minima gets `<slug>.md` at the root; a menu entry pointing at the posts section maps to the theme's blog page;
- the home page (`content/_index.md`) becomes the body of the theme's landing page (al-folio: `_pages/about.md`; Chirpy: `_tabs/about.md`; Minimal Mistakes and minima: `index.md`); an avatar `{{< figure class="avatar" >}}` becomes the theme's profile picture (al-folio profile, Chirpy avatar:, Minimal Mistakes author.avatar);
- site identity from the config (title/author, baseURL, description) is written into the new `_config.yml` (al-folio: `first_name/ last_name`, jekyll-scholar author names; minima: title);
- social profile links found in the Hugo pages (GitHub, LinkedIn, ORCID, Google Scholar, ResearchGate, StackOverflow, ...), the first e-mail address and a link to a CV/resume PDF are written where the theme shows them: al-folio's `_data/socials.yml`, Chirpy's `social: config` block (plus its github/twitter buttons), or Minimal Mistakes' `author: sidebar` links. Posts are not scanned (they may link to other people's profiles), and only profile-shaped URLs count (a GitHub repository link is ignored);
- front matter (YAML --- or TOML +++) is converted to Jekyll YAML: `layout: post` added, `draft: true` becomes `published: false`, `aliases` becomes `redirect_from` (needs the jekyll-redirect-from plugin), Hugo-only keys are dropped;

- `static/` is copied to the site root, so absolute asset URLs like `/images/foo.png` keep working; with `only_referenced = TRUE` only the files actually referenced by the migrated content are copied;
- page-bundle resources (`content/post/my-post/plot.png`) are copied to `assets/img/posts/<post>/` and relative image links are rewritten;
- `{{< figure >}}`, `{{< youtube >}}` and `{{< vimeo >}}` shortcodes are converted; any other shortcode is left in place and reported for manual porting.

The Hugo site is only read, never modified. Content not referenced by the menu, layouts and whatever else cannot be converted is listed in the final report.

The migrated site carries no leftovers from the theme's demo: the scaffold is scrubbed before your content lands, and a final pass removes any demo media (sample photos, videos, audio, notebooks, preview images) that nothing in the migrated site references – what remains is your content plus what Jekyll and the theme need.

Value

Invisibly, a list report: converted posts, drafts, pages, files with remaining shortcodes, config keys set, social links written, static files skipped, and content left for manual migration.

Examples

```
## Not run:
migrate_hugo(
  from = "blog-hugo",      # existing blogdown/Hugo site (never written to)
  to   = "blog-jekyll",    # created here; must not exist yet
  theme = "al-folio",
  only_referenced = TRUE,  # copy only the static files actually used
  theme_color = "red"
)

# then, after bundle_install("blog-jekyll") once, preview locally:
build_site("blog-jekyll")
serve_site("blog-jekyll")

## End(Not run)
```

new_post

Create a new post

Description

Creates `_source/YYYY-MM-DD-slug.Rmd` (default) or `.qmd` (Quarto), or, with `format = "md"`, a plain `_posts/YYYY-MM-DD-slug.md`, all with Jekyll front matter.

Usage

```
new_post(
  title,
  date = Sys.Date(),
  format = c("Rmd", "qmd", "md"),
  tags = NULL,
  categories = NULL,
  dir,
  open = interactive(),
  rmd = NULL
)
```

Arguments

title	Post title.
date	Post date; defaults to today.
format	"Rmd" (R Markdown source, default), "qmd" (Quarto source; rendered by <code>build_site()</code> via the Quarto CLI), or "md" (plain Markdown, written directly to <code>_posts/</code>).
tags, categories	Optional character vectors for the front matter.
dir	Directory in (or under) the site.
open	Open the file (via <code>utils::file.edit()</code>) in interactive sessions?
rmd	Deprecated alias kept for compatibility: <code>rmd = FALSE</code> is <code>format = "md"</code> .

Value

The path to the created file, invisibly.

Examples

```
site <- new_site(tempfile("my-blog"), sample = FALSE)
new_post("Hello world", dir = site, open = FALSE)
new_post("A Quarto post", format = "qmd", dir = site, open = FALSE)
new_post("Plain Markdown, no R code", format = "md", dir = site,
  tags = c("news", "r"), open = FALSE)
list.files(file.path(site, "_source"))
unlink(site, recursive = TRUE)
```

new_site

*Create a new Jekyll site***Description**

Scaffolds a Jekyll site ready for the jekylldown workflow: R Markdown posts live in `_source/`, are knitted to `_posts/*.md`, and figures land in `assets/img/posts/`. Four themes are supported first-class:

Usage

```
new_site(
  dir,
  theme = c("minima", "al-folio", "chirpy", "minimal-mistakes"),
  title = "A jekyll-down site",
  sample = TRUE,
  demo = TRUE
)
```

Arguments

<code>dir</code>	Directory to create the site in. Must be empty or nonexistent.
<code>theme</code>	"minima" (default), "al-folio", "chirpy" or "minimal-mistakes".
<code>title</code>	Site title written to <code>_config.yml</code> (all themes except al-folio, which builds its identity from first/last name – edit its config, or let <code>migrate_hugo()</code> fill it).
<code>sample</code>	Create a sample .Rmd post in <code>_source/</code> ? Default TRUE.
<code>demo</code>	Keep the theme's demo content? Only al-folio ships demo content (sample posts, news, projects, books, an example biography and tens of MB of sample media); default TRUE, set to FALSE for a site that starts empty – the scrub removes the demo pages and any demo media nothing in the site references, so what remains is your content plus what Jekyll and the theme need. <code>migrate_hugo()</code> always starts from a scrubbed site. The other themes start empty regardless.

Details

- "minima" (default) – a minimal self-contained blog, generated locally (no network needed);
- "al-folio" – the academic theme (<https://github.com/alshedivat/al-folio>), fetched from GitHub (a shallow git clone when git is available, the template archive otherwise – git is not required);
- "chirpy" – the technical-blog theme (<https://github.com/cotes2020/jekyll-theme-chirpy>): its starter template is fetched the same way and stripped of the upstream repo tooling, keeping the GitHub Pages deploy workflow;
- "minimal-mistakes" – the general-purpose theme (<https://github.com/mmistakes/minimal-mistakes>), generated locally as a gem-based site (see `set_theme_skin()` for its skins).

On al-folio the feed is rendered from a site-level copy of jekyll-feed's template (see `add_feed()`), so that the theme's title: blank convention yields the author's name as the feed title instead of the literal word "blank".

Value

The normalized site path, invisibly.

Examples

```
# a minimal blog with the locally generated minima theme (no network)
site <- new_site(tempfile("my-blog"))
list.files(site)
```

```

unlink(site, recursive = TRUE)

## Not run:
# the al-folio academic theme, fetched from GitHub, starting empty
new_site("my-site", theme = "al-folio", demo = FALSE)

# the Chirpy starter, or a gem-based Minimal Mistakes site
new_site("my-blog", theme = "chirpy")
new_site("my-blog", theme = "minimal-mistakes", title = "My blog")

## End(Not run)

```

serve_site

Serve the site locally with live reload

Description

Builds the site and serves `_site/` with live reload: keep the preview tab open in the browser and every save of a watched file – `_source/*.Rmd` and `*.qmd`, Markdown pages, `_config.yml`, data files, styles – re-knits what is stale, reruns `jeekyll build` and reloads the page. No manual `build_site()` calls while writing.

Usage

```
serve_site(dir, background = interactive(), port = NULL, ...)
```

Arguments

<code>dir</code>	Directory in (or under) the site.
<code>background</code>	Run the watcher/server in a separate R process? Default TRUE in interactive sessions.
<code>port</code>	Port to serve on; a random free port by default.
<code>...</code>	Passed on to <code>servr::httpd()</code> (e.g. <code>daemon</code> , <code>interval</code>) when <code>background = FALSE</code> .

Details

By default the whole thing runs in a **separate background R process**: your console stays free, nothing is printed there, and rebuild output goes to a log file (`.jeekyll-down-serve.log` in the site; the path is shown when the server starts). Stop it with `stop_server()`. With `background = FALSE` the server runs in this session instead (as a `servr` daemon), which keeps the console usable but briefly busy during each rebuild.

The rebuild is triggered by the open preview tab (it pings the server about once a second); with no browser tab open, nothing rebuilds.

Value

Invisibly, the preview URL.

Rebuild speed

Content-only edits (posts and pages) rebuild with jekyll's incremental mode – only the changed pages are regenerated. Edits with site-wide effects that incremental mode does not track (sass partials, _config.yml, data files) still trigger a full build.

Jekyll is markedly slower on Windows, mostly because real-time antivirus scanning intercepts the thousands of small file reads of every build. If previews feel sluggish, excluding the site folder and jekyll's toolchain directory (tools::R_user_dir("jekyll", "data")) from real-time scanning (Windows Security > Virus & threat protection > Exclusions) makes the single biggest difference. Heavy themes also do less work per build with the demo content removed (new_site(demo = FALSE)).

Examples

```
## Not run:
serve_site("my-site")           # console stays free; open the URL
stop_server("my-site")

## End(Not run)
```

set_element_style	<i>Style one semantic element of the theme</i>
-------------------	--

Description

Styles a single part of the site – the navbar, the footer, headings – without writing CSS: the element name is translated to the theme's own selectors and the properties are appended to the site-local stylesheet in a managed block (one per element; re-running replaces it).

Usage

```
set_element_style(
  element,
  color = NULL,
  background = NULL,
  size = NULL,
  ...,
  dir
)
```

Arguments

element	One of the element names above.
color, background, size	Text color, background color and font-size for the element. Colors accept theme palette names, #hex, or any CSS color.
...	Further CSS properties as named arguments, with _ for -: e.g. font_weight = "600", border_bottom = "none".

`dir` Site root, or any directory inside it. Like `build_site()`, the function climbs to the enclosing site.

Details

Elements: "navbar", "brand" (the site name in the navbar), "socials" (the social icon row – e.g. its size; the sizing also covers what font-size alone would miss: image and SVG icons, and Chirpy's circular buttons, all fixed-size in the themes), "footer", "headings", "post_title", "links", "code", "buttons".

Value

Invisibly, the CSS selector styled.

A word of caution – this is the fragile layer

Unlike `set_theme_style()`, which goes through the CSS variables the theme itself promises, this layer depends on a curated map of theme selectors (.navbar, footer, ...) that upstream can rename at any release. jekylldown checks the installed theme's sass and warns when a mapped selector is no longer found, but the check needs the theme gems installed and cannot catch everything. Prefer `set_theme_style()` whenever a CSS variable covers your case; treat this function as the middle ground before dropping to `add_css()`.

Examples

```
site <- new_site(tempfile("my-blog"))
set_element_style("navbar", background = "#222222", color = "white",
  dir = site)
set_element_style("headings", color = "red", font_weight = "600",
  dir = site)
unlink(site, recursive = TRUE)
```

set_theme_color	<i>Set the al-folio accent color</i>
-----------------	--------------------------------------

Description

al-folio derives its accent (links, buttons, badges) from the `--global-theme-color` CSS variable defined in the theme's `_sass/_themes.scss` – purple in light mode, cyan in dark mode. This function overrides it without touching the theme: the theme's `assets/css/main.scss` is copied into the site (a local file shadows the theme's) and a plain-CSS override block is appended, one rule per selector in which the theme sets `--global-theme-color`, so the override tracks the theme's own light/dark structure. Named colors are resolved from the theme's palette (`$red-color`, `$blue-color`, ... in `_sass/_variables.scss`); hex values are used as-is.

Usage

```
set_theme_color(color = NULL, dir)
```

Arguments

color	A named theme color (e.g. "red", "blue", "green", "cyan", "purple", "pink", "orange", "yellow") or a CSS hex value like "#b71c1c".
dir	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.

Details

The theme files live in the `al_folio_core` gem, so run `bundle_install()` once before calling this. Calling it again replaces the previous override, so the color can be changed at any time.

On a Chirpy site this delegates to `set_theme_style(accent = color)` (the link color); on Minimal Mistakes it points you at `set_theme_skin()` instead.

Value

The hex color applied, invisibly.

Going back to the theme's defaults

`set_theme_color(NULL)` removes the color override. To drop *every* jekylldown customization at once (colors, fonts, element styles, custom CSS), delete the site-local stylesheet (`assets/css/main.scss`; `assets/main.scss` on minima) – it only holds the theme's copy plus the managed blocks, and is recreated on the next customization call.

Examples

```
## Not run:
set_theme_color("red", dir = "my-site")
set_theme_color("#0057b7", dir = "my-site")
set_theme_color(NULL, dir = "my-site")  # back to the theme default

## End(Not run)
```

set_theme_font

Customize the site fonts

Description

Overrides the theme's font family and base size without touching the theme: `@font-face` rules for Google Fonts are inlined into the site-local stylesheet (they are valid anywhere in a stylesheet, unlike `@import`), followed by `font-family`/`font-size` overrides. Re-running replaces the previous override.

Usage

```
set_theme_font(
  family = NULL,
  size = NULL,
  headings = NULL,
  code = NULL,
  google = TRUE,
  dir
)
```

Arguments

family	Body font family (e.g. "Lora"). With google = TRUE the face is fetched from Google Fonts at call time (needs network); otherwise it must be available on the visitor's system.
size	Base font size as a CSS length ("17px", "110%", ...). Themes sized in rem (alfolio, minima) scale the whole site.
headings	Font family for h1-h6 (defaults to family's behavior – i.e. unchanged unless given).
code	Font family for code (code, pre, kbd, samp).
google	Fetch the given families from Google Fonts and inline their @font-face rules? Default TRUE.
dir	Site root, or any directory inside it. Like build_site() , the function climbs to the enclosing site.

Value

Invisibly, the path of the stylesheet written.

Examples

```
# a system font, no download
site <- new_site(tempfile("my-blog"))
set_theme_font("Georgia", size = "17px", google = FALSE, dir = site)
unlink(site, recursive = TRUE)

## Not run:
# Google Fonts, fetched and inlined
set_theme_font("Lora", size = "17px", dir = "my-site")
set_theme_font(code = "JetBrains Mono", dir = "my-site")

## End(Not run)
```

set_theme_skin	<i>Pick a Minimal Mistakes skin</i>
----------------	-------------------------------------

Description

Minimal Mistakes styles the whole site through Sass "skins" compiled into the CSS (there are no CSS variables to override). This sets `minimal_mistakes_skin` in `_config.yml`; the skin list is read from the installed theme gem when available ("default", "air", "aqua", "contrast", "dark", "dirt", "mint", "neon", "plum", "sunrise", ...).

Usage

```
set_theme_skin(skin, dir)
```

Arguments

skin	Skin name.
dir	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.

Value

Invisibly, the skin set.

Examples

```
## Not run:
set_theme_skin("dark", dir = "my-site")

## End(Not run)
```

set_theme_style	<i>Customize theme colors through the theme's own CSS variables</i>
-----------------	---

Description

al-folio and Chirpy define their palettes as CSS variables, with one block per color scheme (light and dark). This function overrides those variables from R – the most robust customization layer, because the theme's components all read from the variables, and light/dark mode keeps working.

Usage

```
set_theme_style(..., dir)
```

Arguments

...	Named style options from the list above.
dir	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.

Details

Every option takes either a single color (applied to both modes) or a vector like `c(light = "#ffffff", dark = "#1c1c1d")`. Colors can be a #hex value or any CSS color (on al-folio, also a theme palette name like "red" or "blue"). Re-running replaces the previous override.

Options on **al-folio**: accent and hover (the accent color – `set_theme_color()` is shorthand for both), background, text, text_light, divider, highlight, code_background, card_background, footer_background, footer_text, footer_link.

Options on **Chirpy**: accent (the link color), background, text, text_light, heading, divider, code_background, card_background, sidebar_background, sidebar_active.

Minimal Mistakes has no CSS variables (its palette is compiled from Sass skins): use `set_theme_skin()` to switch skins, and `set_element_style()/add_css()` for finer control. The same applies to min-ima.

This needs the theme gems: run `bundle_install()` once first.

Value

Invisibly, the named list of CSS variables written.

Examples

```
## Not run:
set_theme_style(
  accent = "red",
  footer_background = "#222222",
  background = c(light = "#fffd7", dark = "#1c1c1d"),
  dir = "my-site"
)

## End(Not run)
```

stop_server

Stop servers started by `serve_site()`

Description

Stops the background serve process of dir (found through the `.jekyll-down-serve.pid` file the server wrote), plus any in-session servr daemons.

Usage

```
stop_server(dir)
```

Arguments

dir Directory in (or under) a site served with background = TRUE; NULL only stops in-session daemons.

Value

NULL, invisibly.

Examples

```
## Not run:
stop_server("my-site")

## End(Not run)
```

use_pages_workflow	<i>Add a GitHub Pages deploy workflow to the site</i>
--------------------	---

Description

Writes the standard GitHub Actions workflow that builds the site with Jekyll and publishes it to GitHub Pages on every push (.github/workflows/jekyll.yml): checkout, Ruby with cached bundle, jekyll build, upload, deploy. After pushing, enable it once in the repository settings: *Settings > Pages > Source: GitHub Actions*.

Usage

```
use_pages_workflow(dir)
```

Arguments

dir Site root.

Details

al-folio and Chirpy already ship their own deploy workflow – for those sites this function leaves it alone and tells you so. It is meant for the themes that do not (minima, Minimal Mistakes, most others). Mind the Pages source for the bundled ones: Chirpy’s workflow publishes like this one (*Source: GitHub Actions*), but al-folio’s pushes the built site to a gh-pages branch, so its Pages source must be *Deploy from a branch > gh-pages*.

Note the division of labor: the workflow only runs jekyll build. Knitting stays local – run `build_site()` (with `local_jekyll = FALSE` if you have no local Jekyll), then commit both `_source/` and the generated `_posts/*.md` and push.

The workflow caches gems, which requires a committed Gemfile.lock: run `bundle_install()` once and commit the lockfile it produces.

Value

Invisibly, the path of the workflow file (or of the theme's own workflow when one already exists).

Examples

```
site <- new_site(tempfile("my-blog"))
use_pages_workflow(site)
unlink(site, recursive = TRUE)
```

use_r_bloggers	<i>Set a site up for R-Bloggers</i>
----------------	-------------------------------------

Description

R-Bloggers (<https://www.r-bloggers.com/>) aggregates R posts from a full-text feed that contains R content only, and asks for a link back to it on the blog. This function does the site side of that setup in one call. It writes an R-only feed at `feed/<category>.xml` through `add_feed()`, which also fixes the feed title on al-folio sites. On al-folio it adds a line under the blog header with links to the category, to R-Bloggers and to the feed. That line is a marker-delimited block in `_pages/blog.md`, replaced on re-runs; a link to R-Bloggers you wrote there yourself is left alone. On other themes the HTML snippet is printed for you to place. Finally it prints the feed URL to submit at <https://www.r-bloggers.com/add-your-blog/> once the site is published.

Usage

```
use_r_bloggers(category = "R", dir)
```

Arguments

<code>category</code>	The category that marks R posts; "R" by default.
<code>dir</code>	Site root, or any directory inside it. Like <code>build_site()</code> , the function climbs to the enclosing site.

Details

Only posts whose front matter carries the category enter the feed: `new_post("...", categories = "R")`, or `categories: R` by hand. Posts about anything else stay out, which is what R-Bloggers asks.

Value

Invisibly, a list with `feed` (the feed page's path), `url` (the public feed URL, from the site's `url` and `baseurl`) and `link` (the page the link was added to, or `NULL`).

Examples

```
site <- new_site(tempfile("my-blog"))
use_r_bloggers(dir = site)
unlink(site, recursive = TRUE)
```

Index

add_css, [2](#)
add_css(), [19](#), [23](#)
add_feed, [3](#)
add_feed(), [16](#), [25](#)
add_footer_credit, [4](#)
add_mathjax, [5](#)

build_site, [6](#)
build_site(), [3–5](#), [7](#), [9–11](#), [15](#), [17](#), [19–25](#)
bundle_install, [7](#)
bundle_install(), [5](#), [8](#), [13](#), [20](#), [23](#), [24](#)

check, [8](#)
check(), [9](#), [10](#)

install_git, [8](#)
install_git(), [7](#)
install_quarto, [9](#)
install_quarto(), [11](#)
install_ruby, [10](#)
install_ruby(), [9](#)

knit_post, [11](#)
knit_post(), [4](#), [6](#)
knitr::render_jekyll(), [11](#)

migrate_hugo, [12](#)
migrate_hugo(), [4](#), [16](#)

new_post, [14](#)
new_post(), [9](#)
new_site, [15](#)
new_site(), [4](#), [12](#)

processx::run(), [7](#)

remove_footer_credit
 (add_footer_credit), [4](#)
remove_footer_credit(), [5](#)

serve_site, [17](#)

serve_site(), [7](#), [9–11](#), [23](#)
servr::http(), [17](#)
set_element_style, [18](#)
set_element_style(), [2](#), [23](#)
set_theme_color, [19](#)
set_theme_color(), [13](#), [23](#)
set_theme_font, [20](#)
set_theme_font(), [2](#)
set_theme_skin, [22](#)
set_theme_skin(), [16](#), [20](#), [23](#)
set_theme_style, [22](#)
set_theme_style(), [2](#), [19](#)
stop_server, [23](#)
stop_server(), [17](#)

use_pages_workflow, [24](#)
use_r_bloggers, [25](#)
use_r_bloggers(), [3](#), [4](#)
utils::file.edit(), [15](#)